System-Level Test Design

Testability Assessment

- Controllability: **CONCERNS**
  - Greenfield system, so we control most seams (API/service boundaries, matching algorithm, storage repository layer).
  - Key controllability risks are (a) triggering scheduled matching deterministically and (b) authenticating in local + CI environments when using SWA EasyAuth.
  - Mitigation: design explicit “test seams” (see “Testability Concerns”) so we can seed data, run a single match cycle, and simulate authenticated users without production-only hacks.
- Observability: **CONCERNS**
  - Architecture calls for structured logs, system-event logging (system actions only), correlation IDs, and delivery logging for notifications.
  - Risk: “All-free MVP” decisions (no App Insights by default) reduce observability unless logs are made easy to access and consistent across local/staging/prod.
  - Mitigation: enforce structured logging + correlation IDs from day 1, and include contract tests asserting error shapes and trace/correlation headers where applicable.
- Reliability: **PASS (with guardrails)**
  - Scheduled matching + idempotency keys + atomic cycle processing is explicitly required; Table Storage partitioning by `ProgrammeId` provides a strong backbone for deterministic access patterns.
  - Risk concentrates in timer-trigger execution behaviour (retries, duplicates) and eventual consistency in storage/email delivery.
  - Mitigation: test idempotency and failure modes as P0/P1 at integration level.

Architecturally Significant Requirements (ASRs)

Risk scoring uses Probability (1–3) × Impact (1–3). Scores **≥6** require mitigation before release gates.

ASR ID	Category	Requirement (evidence)	Probability	Impact	Score	Test Strategy (primary level)
ASR-001	SEC/DATA	**No cross-programme data leakage** (partitioning by `ProgrammeId`, “avoid cross-partition scans”, repository enforcement)	2	3	6	Integration + negative contract tests
ASR-002	SEC	**Auth/authz enforced per Function; claims treated as untrusted** (SWA EasyAuth + explicit checks per handler)	2	3	6	Contract + integration tests with forged/empty claims

ASR ID	Category	Requirement (evidence)	Probability	Impact	Score	Test Strategy (primary level)
ASR-003	BUS/SEC	**Trust contract: no individual-level reporting, no surveillance analytics**	2	3	6	Contract tests for role endpoints + “trust-guard” suite
ASR-004	OPS/TECH	**Matching is atomic and idempotent** (keyed by `ProgrammeId + CycleId/CycleDate`)	2	3	6	Integration tests + deterministic matching unit tests
ASR-005	OPS	**No silent errors / no partial matches** (clear failure behaviour)	2	3	6	Integration tests for failure injection + log assertions
ASR-006	OPS	**Email delivery logged and retried; sent within 60 minutes**	2	2	4	Integration tests around notification pipeline
ASR-007	DATA/SEC	**Deletion/retention honoured** (delete request window, minimal audit retention)	2	3	6	Integration tests + scheduled job tests
ASR-008	PERF	**Basic web performance: sub-3s load** (PRD NFR)	2	2	4	E2E smoke perf budget + Lighthouse (CI)
ASR-009	REL	**Best-effort reliability, idempotent retries** (serverless timer)	2	2	4	Integration tests around retry/idempotency
ASR-010	A11Y	**WCAG 2.1 AA aspiration; keyboard support**	2	2	4	Component + E2E accessibility checks

Test Levels Strategy

Target coverage distribution for MVP (risk-based):

- Unit: **~60%** — matching algorithm, idempotency key derivation, DTO validation helpers, permission predicates, and pure transformations.
- Integration: **~30%** — API handlers + repositories against a test storage backend, email sending abstraction, “run matching cycle” service with deterministic inputs.
- E2E: **~10%** — minimal set of critical user journeys (join/pause/leave, match view, transparency page) and trust-critical UI assertions.

Guardrail: avoid duplicating the same behaviour across E2E and API tests unless it is a trust- or security-critical invariant that benefits from defence-in-depth.

NFR Testing Approach

- Security
  - Contract/integration tests for unauthenticated vs authenticated access, role checks, and "claims are untrusted" behaviour (missing/invalid claims should fail safely).
  - Negative contract tests for cross-programme access attempts, admin/owner endpoints returning only aggregate data, and minimum-N thresholds on aggregates.
  - Dependency + config scanning in CI (exact tool choice can be finalised during `*ci` workflow; focus is "no secrets, no PII in logs").
- Performance
  - Establish a minimal baseline: SPA cold load under a defined environment, API p95 latency budget for core endpoints (join, match view).
  - Lightweight load checks for the matching job and notification pipeline (enough to surface obvious regressions; no heavy-scale proof needed for MVP).
- Reliability
  - Integration tests for idempotent matching: repeat the same cycle input and assert no duplicate matches/notifications are produced.
  - Failure-injection tests: storage transient failures, email send failure paths, and timer "double fire" behaviour.
  - Contract tests for Problem Details and "no silent error" behaviour (errors are visible and diagnosable without exposing sensitive data).
- Maintainability
  - Enforce the project's test quality definition of done: deterministic waits, parallel-safe fixtures, no hard waits, fast-running suites.
  - CI runs are sliced by priority/tags (`@p0`, `@p1`) to keep feedback loops short while preserving a full nightly run.

Test Environment Requirements

Local developer environment

- SWA local runtime (to reproduce auth + route behaviour) + Azure Functions Core Tools.
- Storage test backend:
  - Preferred: Azurite (Table support if/where available), otherwise a dedicated test storage account.
- Deterministic test execution:
  - Seedable clock and RNG for matching.
  - Ability to run a single matching cycle on-demand (dev/test-only seam).

CI environment (GitHub Actions)

- Build + unit tests for `apps/web` and `apps/api`.
- Contract tests for API shapes (including ISO 8601 and Problem Details).
- E2E smoke (Playwright) against a locally hosted app+api bundle or a deployed staging environment.
- Isolation:

  - Dedicated storage namespace per run (unique table names or a unique storage account/container prefix).
  - Email sender must be stubbed or routed to a safe sink (no real recipients).

Staging environment (recommended for E2E confidence)

- A staging deployment with:
  - Test Entra users (or a controlled auth simulation strategy).
  - Realistic storage + queue/timer scheduling.
  - Safe email sink (e.g., allowlisted recipients only) and delivery logging enabled.

Testability Concerns (if any)

1. **SWA EasyAuth in automated tests**
   - Risk: browser E2E tests can get stuck on real Entra login flows; API tests need a reliable way to represent claims.
   - Recommendation: design an explicit test-auth strategy early (e.g., local proxy that injects SWA client principal headers for test runs, and contract tests that cover missing/invalid claim headers).
2. **Timer-trigger matching control**
   - Risk: scheduled triggers are hard to reproduce and can be flaky in CI.
   - Recommendation: keep timer trigger thin and move matching into an injectable service; add an internal/dev-only trigger to run a cycle deterministically for tests.
3. **Email delivery verification without leaking PII**
   - Risk: verifying notification content encourages capturing email bodies/logs (PII) in test artefacts.
   - Recommendation: test via a stubbed sender that records only non-identifying metadata + template IDs, and explicitly forbid logging recipient bodies.
4. **Anti-surveillance constraints as "negative requirements"**
   - Risk: it's easy to accidentally add endpoints/log fields that enable individual-level reporting.
   - Recommendation: add a "trust-guard" suite of negative tests that explicitly prove "this data cannot be retrieved".

Recommendations for Sprint 0

1. Run `*framework` to scaffold the test harness (Playwright E2E structure, fixture patterns, tagging conventions).
2. Run `*ci` to establish quality gates (unit + contract + smoke E2E) with fast feedback loops.
3. Implement the **trust-guard suite** early:
   - Contract tests proving admin/owner visibility stays aggregate-only.
   - Negative tests proving no cross-programme access paths.
4. Implement deterministic seams up front:
   - Seedable time + RNG for matching.
   - A clear, testable idempotency key strategy (`ProgrammeId + CycleDate` in programme time zone).
5. Establish email testing discipline:
   - Stub sender in CI by default; require explicit opt-in to send to allowlisted addresses.
   - Delivery logging contract tested (attempt/sent/fail) without bodies.